IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Serial No. 10/700,254
Confirmation No. 3682

I hereby certify that this correspondence is being transmitted to the United States Patent & Trademark Office via electronic submission or facsimile on the date indicated below:

04/08/2009 /Pamela Gerik/
Date Pamela Gerik

APPEAL BRIEF

Dear Sir:

Further to the Notice of Appeal filed December 10 2008, Appellant presents this Appeal Brief. Appellant hereby appeals to the Board of Patent Appeals and Interferences from the final rejection of pending claims 1-25 and respectfully requests that this appeal be considered by the Board.

I. REAL PARTY IN INTEREST

The subject application is owned by Ramal Acquisition Corp. as evidenced by the document recorded at reel 018779 and frame 0519. Ramal Acquisistion Corp. is a wholly owned subsidiary of Pervasive Software, Inc.

II. RELATED APPEALS AND INTERFERENCES

No appeals, interferences, or judicial proceedings are known which would directly affect or be directly affected by or have a bearing on the Board's decision in this appeal. A related case filed by the same assignee is pending on appeal (Application No. 10/700,152) but the subject matter of the claims and applicable art raise different issues.

III. STATUS OF THE CLAIMS

Claims 1 -25 are pending in the application and stand finally rejected.

IV. STATUS OF AMENDMENTS

No claim amendments were filed subsequent to their final rejection. The Appendix hereto reflects the current state of the claims.

V. SUMMARY OF CLAIMED SUBJECT MATTER

Independent claim 1 recites a system for managing data utilizing one or more processors and a single operating system comprising: a plurality of map components, each map component having one or more ports for accepting data and for producing data and each map component encapsulating a particular dataflow pattern (Specification -- pg. 7, lines 15-18; pg. 12, lines 26-29; pg. 13, lines 7-11; pg. 15, lines 1-5); compiler tools for organizing and linking said map components using said ports into an executable dataflow application (Specification -- pg. 7, lines 17-20); and an executor for creating and managing data communication among map components in the dataflow application (Specification -- pg. 7, lines 20-24; pg. 11, lines 28-31; pg. 16, lines 14-22; pg. 29, lines 16-21)and executing the dataflow application on said one or more processors in parallel with each map component as a separate thread of execution with data supplied to the system (Specification -- pg. 8, lines 12-14; pg. 29, lines 18-22; pg. 30, lines 6-11).

Independent claim 16 recites a method of transforming data in a parallel processing environment comprising a single operating system and one or more processors wherein: map components are assembled visually into an integrated dataflow application by linking map processes within map components (Specification -- pg. 8, lines 10-14); the integrated dataflow application is executed in parallel by recognizing the linked map processes within the map components and allocating a thread to each map process (Specification -- pg. 8, lines 12-15); and each map process is executed on its allocated thread substantially in parallel, and said data

resides in memory accessible to each map process (Specification -- pg. 16, lines 14-18; pg. 29, lines 18-22; pg. 30, lines 6-11).

Independent claim 18 recites a method of managing data comprising: accessing a library of map components at least some of said map components constituting a specific data transformation and having input and output ports (Specification -- pg. 8, lines 1-4; pg. 13, lines 10-15; pg. 21, lines 29-32); assembling a dataflow application using map components from said library linked together using said ports (Specification -- pg. 8, lines 3-5; pg. 16, lines 1-5; pg. 22, lines 20-25; Fig. 5); and executing the assembled dataflow application with source data by assigning a thread to each map component where said threads execute in parallel on said source data without data partitioning (Specification -- pg. 7, lines 18-23; pg. 8, lines 5-10; pg. 29, lines 18-22; pg. 30, lines 6-11; Fig. 14).

Dependent claim 2 recites the system of claim 1, the compiler including tools for visually creating composite components comprising other map components and tools for visually assembling map components into a dataflow application (Specification -- pg. 8, lines 10-14; pg. 19, lines 29-31).

Dependent claim 7 recites the system of claim 1, at least one of said ports linked to transfer specific types of data (Specification -- pg. 7, lines 25-30; pg. 41, lines 18-28).

Dependent claim 9 recites the system of claim 1, at least one of said ports being composite, comprising a plurality of hierarchical ports (Specification -- pg. 7, lines 22-28; pg. 18, lines 24-30; pg. 42, lines 13-30; Fig. 20).

Dependent claim 10 recites the system of claim 1, at least one of said ports supporting multi-valued null data tokens (Specification -- pg. 7, lines 25 – pg. 8, line 8).

Dependent claim 12 recites the system of claim 1, at least one of said map components being composite comprising a number of hierarchical dataflow graphs (Specification -- pg. 7, lines 22-28; pg. 42, lines 13-30; Fig. 20).

Dependent claim 13 recites the system of claim 1, the compiler operating to remove design time links between map components to produce a flat dataflow graph containing a plurality of map processes for execution (Specification -- pg. 16, lines 1-5; pg. 26, lines 1-5; pg. 43, lines 5-31; Figs. 24, 26).

Dependent claim 14 recites the system of claim 1, the executor operating to assign a thread to each map process for parallel execution (Specification -- pg. 16, lines 14-24; pg. 21, lines 20-27; pg. 29, lines 16-31).

Dependent claim 17 recites the method of claim 16, wherein a plurality of map processes read data tokens from input ports and write data tokens to output ports (Specification -- pg. 30, lines 6-20; pg. 31, lines 7-14; pg. 33, lines 6-10).

Dependent claim 20 recites the method of claim 18, the map components including polymorphic ports which declare status as input and output ports during assemblage (Specification -- pg. 15, lines 16-19).

Dependent claim 24 recites the method of claim 16, communication between said processes executing in parallel being managed by an executor separate from the operating system (Specification -- pg. 7, lines 19-24; pg. 16, lines 14-20; pg. 48, lines 1-7; Fig. 29).

Dependent claim 25 recites the method of claim 18, including: determining if a port will block execution of a thread (Specification -- pg. 34, lines 5-31); and avoiding a deadlock by allowing the data queue to grow at said determined port (Specification -- pg. 34, lines 5-31).

Data processing in a business process takes a variety of forms, such as data warehousing, decision support software, analytical software, customer relationship management. Such data processing invariably involves transforming the data for use.

Dataflow graphs are widely recognized as an effective means to specify data processing software. Their value lies in the succinct presentation and explicit definition of data transfers as the data progresses through a chain of transformation processes.

The present invention relates to computationally efficient data transformation and management applications using dataflow graphs particularly suited for parallel computation where each map process of a dataflow application is executed as a separate thread of execution. The system and methods described herein center on the generation of an execution plan for a single system with one or more computer cores (processors), where the plan executes in parallel as single operating system process on multiple threads.

Specific embodiments contemplates a method for developing a dataflow application where one or more data transformations are developed by accessing a library of map components (Specification -- pg. 21, lines 28-31; Fig. 1). That is, data transformations having ports are assembled into a map component with links between ports. Ports may be hierarchical (Specification -- pg. 25, lines 6-11). One or more map components are compiled with syntactic and semantic analysis (Specification -- pg. 15, lines 10-15; pg. 26, lines 1-5). Compiled map components are synthesized into an executable dataflow application with threads assigned to each map component. Certain embodiments support multi-valued null data tokens (Specification -- pg. 31). Other embodiments describe deadlock detections (Specification -- pp. 32-35).

No means plus function or step plus function type claims are presented.

VI. GROUNDS OF REJECTION TO BE REVIEWED

1. Claims 1 - 25 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over U.S. Patent No. 6,330,008 et. al (hereinafter "Razdow") in view of U.S. Patent No. 5,499,349 Nikhil et al. (hereinafter "Nikhil").

VII. ARGUMENT

1. Rejection of Claims 1 - 25 under 35 U.S.C. § 103(a)

Claims 1 - 25 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Razdow in view of Nikhil. The factual inquiries set forth in *Graham v. John Deere Co.,* 383 U.S. 1, 148 U.S.P.Q. 459 (1966), establish the background for determining obviousness under 35 U.S.C. §103, see also, MPEP 706 and 2141. The primary considerations are: the scope and content of the prior art; the differences between the prior art and the claims in issue; the level of ordinary skill in the pertinent art; and secondary considerations, such as commercial success, long felt and unresolved needs, failures of others, teaching away, etc. See also, *KSR Int'l. Co., v. Teleflex, Inc.,* No. 04-1350 (U.S. Apr. 30, 2007). While the *KSR* court rejected the rigid application of the "teaching, suggestion, motivation" test, the reason a person of ordinary skill in the relevant field would combine prior elements must be made explicit. *KSR*, slip op. at 14. The Patent Office in its post *KSR* Examination Guidelines now looks to seven (7) indicia to support an obviousness rejection. In this case, applicable indicia must show either 1) known methods were combined to yield predictable results or, 2) there exists some teaching, suggestion, or motivation within the references to combine the references to arrive at the claimed invention.

A. The scope and content of the prior art

Razdow describes a method for describing a record-oriented dataflow process (as a single dataflow graph) in a design environment and for executing the dataflow process using both horizontal partitioning and pipeline parallelism. In both cases, the parallelism is at the operating system process level and requires the use of interprocess communications (IPC) mechanisms both to control and transfer data between processes.

In Razdow, the compiler tools are used to generate the scores for dataflows with parallel operators. This is essentially a process of generating the dataflow graphs needed to effect the execution of the original dataflow graph on different computer processors (nodes) where each processor is responsible for processing a single, horizontal partition of the input data. That is, Razdow executes the dataflow graph by partitioning the processes to separate computers on a network of computers, where each computer has its own operating system. I.e. the focus is on

producing the score that makes it possible to run multiple instances of the same dataflow graph in parallel on multiple computer systems.

Razdow does not assign threads for each operator or process in the dataflow graph. Instead the same dataflow graph is executed on one or more computer processors in a network of computers and the data is partitioned. On each of these processors, the operators are forked as separate operating system (OS) level processes, requiring OS level interprocess communications (IPC) mechanisms for transferring data between operators.

Razdow discloses a system for visualizing and monitoring the execution of a dataflow application across multiple computers by generating an execution plan that partitions the work across multiple computers where each computer executes the same execution plan on a subset of the input data. The data is partitioned into subsets and allocated to a particular computer.

Nikhil does disclose the use of "threads" but does not disclose or suggest the scheme or architecture described in the present application. Nikhil describes a computer processor architecture. The processor provides instruction pipelining and it is machine instructions that are enqueued. This is in contrast to the present invention which is concerned with data pipelining.

Nikhil uses a different threading model using the fork and join instructions. When the processor encounters a fork instruction on a thread, processing is split into two threads to execute instructions in parallel. The two threads are joined back together. Threads are created and destroyed as fork and join instructions are encountered in the machine code. This approach is instruction driven, as opposed to the data driven approach of the present application. In the present application, a thread is dedicated to a complex operation – "map component" operator. There is no "fork and join." Therefore, while Nikhil can execute data flow graphs (Fig. 11), the method of execution is different and concerned with processing machine instructions efficiently.

B. Differences between the prior art and claims at issue

i. Independent Claims 1 and 16.

The final office action takes the position that all of the elements of claims 1 and 16 are shown in Razdow, except Razdow is silent about "with each map component as a separate thread of execution." While it is certainly correct that Razdow is silent as to certain passages of the last element of claim 1, other distinctions exist between the language of claim 1 and Razdow.

First, the compiler tools in Razdow are used to generate the scores for dataflows with parallel operators. This essentially horizontally partitions the data so that each node (or processor) can execute a single data partition in parallel. That is, these partitions are used to execute the original dataflow graph with horizontally partitioned data. The present invention does not horizontally partition the data, but rather the compiler transforms a hierachical composition of dataflow graphs into a single flattened dataflow graph for execution. In the example of claim 1, the "compiler" organizes and links map components into an "executable dataflow application." That is not true with Razdow, see e.g., col. 2, lines 45-55 and Fig. 6 describing data partitioning.

Second, Razdow is describing a system for monitoring the performance of the execution of parallel dataflow graphs. The parallelism is at the operating system level and requires the use of interprocess communication to control and transfer data between parallel processes (Razdow at col. 2, lines 45-55; col. 10, lines 5-20). In contrast, the present application addresses a single instance of a dataflow application executed in parallel by running each operator (map process) of the dataflow graph on a separate thread of execution as part of a single operating system process. As called for in claim 1, the compiler creates "an executable dataflow application" which is then executed by the "executor." In claim 1, the "executor" manages the data communications among map components (not true in Razdow) and "each map component" is executed in "parallel" as "a separate thread of execution" while executing this single dataflow application.

The executor of claim 1 finds no couterpart in Razdow, and indeed claim 1 calls for execution of the dataflow application in parallel with each map component at least as a separate thread. (Claim 1 allows for the possibility that several threads could be assigned to several processes in a single map component). Razdow do not assign processes or operators to threads.

While it is true Nikhil mentions multithreaded applications and the ability to process "dataflow applications" (Fig. 11), the architecture of Razdow and purposes are completely inopposite to the present invention. Thus, parellizing the multiprocesses of Razdow as "threads" (from Nikhil) is still just a horizontal partitioning of data to execute the original dataflow graph. Thus, even if Razdow and Nikhil were hypothetically combined, such combination would not meet the limitations of claim 1.

Given the architecture of Razdow, there is no teaching, suggestion, or motivation to combine Nikhil with Razdow. That is, Nikhil would only gloss onto the architecture of Razdow a processing element that could fork into 2 threads. The resulting combination still is a horizontal data partitioning schema. Similarly, while the methods of Razdow and Nikhil are "known methods" the "predictable results" do not arrive at the claim limitations of claim 1.

The above observations are equally applicable to independent claim 16. To recount, Razdow discloses a system for executing a dataflow application across multiple processors by generating an execution plan that partitions the work – i.e. each processor executes the same execution plan on a subset of the data. In contrast, the present application addresses an architecture to generate a single execution plan for a single system that executes in parallel as a single OS process using threads. That is Razdow does not assign an operator for each operator in a dataflow graph. Even if Razdow were modified with Nikhil, the result would not assign a thread to each operator in a dataflow graph. Therefore, as in claim 16, a single "integrated dataflow application is executed" where a thread is allocated to each map process. While Razdow relies on interprocess communication to control and transfer data, in the present invention a shared memory is used to transfer data between map processes. As recited in claim 16, "said data resides in memory accessible to each map process."

Independent claim 16 addresses a preferred method in the context of a parallel processing environment comprising a single operating system and one or more processors and to emphasize that "each map process is executed on its allocated thread substantially in parallel, and said data resides in memory accessible to each map process." Razdow does not assign threads for each operator or process in the dataflow graph. Instead the same dataflow graph is executed on one or more computer processors in a network of computers with IPC coordination. A hypothetical modification of Razdow with Nikhil does not address or suggest the claim limitation of claim 16.

ii. Independent Claim 18

The above observations are equally applicable to claim 18 and incorporated by reference. In addition, claim 18 calls for "accessing a library of map components" where some of the map components constitute a specific data transformation. The Office Action cites Figs. 6 and 11 and accompanying text, as well as col. 2, lines 45-55.

Independent claim 18 emphasizes that one method in accordance with the present invention assigns a thread to each operator (map component) where the threads execute in parallel without the data partitioning found in Razdow. Even if Razdow were modified by Nikhil, the same dataflow graph would be executed on one or more computer processors in a network of computers or as a separate thread of execution on partitioned data..

iii. Claims 2 and 13

Claims 2 and 13 relate to different features of the compiler of claim 1 and the arguments above with respect to claim 1 are applicable and incorporated herein. While Razdow does provide visual tools for monitoring performance of parallel dataflow applications, Razdow does not provide, either alone or in combination with Nikhil, compiler tools "for visually creating composite components ..." Similarly, in relation to claim 13, Razdow does not "remove design time links between map components to produce a flat dataflow graph containing a plurality of map processes for execution." The passage recited in the Office Action (Col. 2, lines 45-55 and Fig. 6) disclose a "substitution" process, but the focus is on producing the score that makes it possible for Razdow to run multiple instances of the same dataflow graph in parallel on multiple

computer systems. The invention of claim 13 is a flattening process where each map process is run on a separate thread.

iv. **Claims 14 and 24**

Claims 14 and 24 relate to different features of the "executor" of claims 1 and 16 respectively, and the arguments above with respect to claims 1 and 16 are applicable and incorporated herein. Razdow does not provide, either alone or in combination with Nikhil, an executor "operating to assign a thread to each map process for parallel execution" as called for in claim 14. Razdow executes the same dataflow graph on one or more computer processors where the data is partitioned. Similarly, claim 24 calls for an embodiment where the executor manages the communication between processes separate for the operating system. This is in contrast to the IPC schema of Razdow.

v. **Claims 7, 9, 10, 12, 17, 20, and 25**

Claims 7, 9, 10, 12, 17, 20, and 25 relate to different features and operation of the "ports" of claims 1, 16, and 18, and the arguments above with respect to independent claims 1, 16, and 18 are applicable and incorporated herein. Specifically, claims 7, 9, and 12 describe the composite and hierarchical nature of the ports. Razdow does describe simple, scalar types of ports bsed on record-oriented dataflow. In claims 7, 9 and 12, the ports can additionally include object types and hierarchical (i.e. composite) types of ports. (Not disclosed or suggested by Razdow or Nikhil.) Indeed, as mentioned in the preferred embodiment the data is not moved along the links as records, but rather as streams of scalar tokens. Closely related are the limitations of claim 20, which calls for a type of polymorphic ports. While the Office Action cites Figs. 11 and 15 of Razdow for support, there does not appear to be any disclosure of "polymorphic ports" as understood in the present invention.

Claims 10 and 17 relate to ports that support the use of null data tokens. As described in the present application, the use of multi-valued null data tokens allow downstream operators to distinguish between undefined values and upstream error conditions. The Office Action cites to Fig. 15 and Col. 8, lines 50 - 55 of Razdow, but there does not appear to be a disclosure of the data tokens as called for in claims 10 and 17.

Claim 25 adds to claim 18 an embodiment for "deadlock" avoidance. The cited passage of Razdow (Col. 3 and Fig. 3) does not support an inference of the limitations of claim 25.

Dependent claim 11 adds support for encrypted components, so that users can develop and redistribute interesting component assemblies without fear of disclosing the implementation of the component assembly.

* * *

For the foregoing reasons, it is submitted that the Examiner's rejection of pending claims 1-25 was erroneous, and reversal of the Examiner's decision is respectfully requested.

Respectfully Submitted,

/Charles D. Huston/
Charles D. Huston
Registration No. 31,027
Attorney for Appellant

Customer No. 35617
Date: April 8, 2009

VIII. APPENDIX

The present claims on appeal are as follows.

1. A system of managing data utilizing one or more processors and a single operating system, comprising:

a plurality of map components, each map component having one or more ports for accepting data and for producing data and each map component encapsulating a particular dataflow pattern;

compiler tools for organizing and linking said map components using said ports into an executable dataflow application; and

an executor for creating and managing data communication among map components in the dataflow application and executing the dataflow application on said one or more processors in parallel with each map component as a separate thread of execution with data supplied to the system.

2. The system of claim 1, the compiler including tools for visually creating composite components comprising other map components and tools for visually assembling map components into a dataflow application.

3. The system of claim 1, at least one map component having properties determining map component design behavior.

4. The system of claim 1, at least one map component having properties that affect map component execution behavior.

5. The system of claim 1, at least one of the map components comprising a composite component encapsulating a particular dataflow pattern using other map components as subcomponents.

6. The system of claim 1, at least one of the map components comprising a scalar map component to process a specific data transformation.

7. The system of claim 1, at least one of said ports linked to transfer specific types of data.

8. The system of claim 1, at least one of said ports initially defined as a generic port for processing generic types of data, said generic port being later synthesized to transfer a specific sub-type of data.

9. The system of claim 1, at least one of said ports being composite, comprising a plurality of hierarchical ports.

10. The system of claim 1, at least one of said ports supporting multi-valued null data tokens.

11. The system of claim 1, at least one of said map components being encoded as an encrypted extensible markup language (XML) document.

12. The system of claim 1, at least one of said map components being composite comprising a number of hierarchical dataflow graphs.

13. The system of claim 1, the compiler operating to remove design time links between map components to produce a flat dataflow graph containing a plurality of map processes for execution.

14. The system of claim 1, the executor operating to assign a thread to each map process for parallel execution.

15. The system of claim 1, the compiler tools operating to perform syntactic and semantic analysis, type inference and validation.

16. A method of transforming data in a parallel processing environment comprising a single operating system and one or more processors wherein:

map components are assembled visually into an integrated dataflow application by linking map processes within map components;

the integrated dataflow application is executed in parallel by recognizing the linked map processes within the map components and allocating a thread to each map process; and

each map process is executed on its allocated thread substantially in parallel, and said data resides in memory accessible to each map process.

17. The method of claim 16, wherein a plurality of map processes read data tokens from input ports and write data tokens to output ports.

18. A method of managing data comprising:

accessing a library of map components at least some of said map components constituting a specific data transformation and having input and output ports;

assembling a dataflow application using map components from said library linked together using said ports; and

executing the assembled dataflow application with source data by assigning a thread to each map component where said threads execute in parallel on said source data without data partitioning.

19. The method of claim 18, including imposing properties on the map components during assembly constraining the assemblage of the dataflow application.

20. The method of claim 18, the map components including polymorphic ports which declare status as input and output ports during assemblage.

21. The system of claim 14, the executor operating on a single CPU in a hyperthread architecture.

22. The system of claim 14, the executor operating on a multiple processor core with at least some threads assigned to different processors.

23. The system of claim 14, the executor operating on multiple processors in a distributed network configuration.

24. The method of claim 16, communication between said processes executing in parallel being managed by an executor separate from the operating system.

25. The method of claim 18, including:

determining if a port will block execution of a thread; and

avoiding a deadlock by allowing the data queue to grow at said determined port.

IX. EVIDENCE APPENDIX

There is no evidence of record in the present application.

X. RELATED APPEALS APPENDIX

No appeals, interferences, or judicial proceedings are known which would directly affect or be directly affected by or have a bearing on the Board's decision in this appeal. A related case filed by the same assignee is pending on appeal (Application No. 10/700,152) but the subject matter of the claims and applicable art raise different issues.